8Heisenberg Model Quantum Material Simulation Software

User’s Guide and Documentation

### Physics Background

### System Requirements

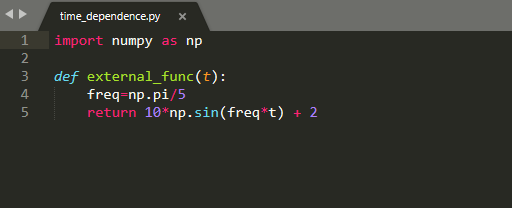
This software requires the most recent version of:

1. IBM’s Qiskit package
   1. Qiskit is IBM’s quantum computing software development framework. This code interfaces with IBM quantum simulators and quantum hardware.
   2. See [Qiskit’s documentation here](https://qiskit.org/documentation/)
2. Matplotlib
   1. This is only needed if plotting is desired (see plot parameter), so it will only be imported if this parameter is set to “y”
3. Numpy
   1. Numpy is used extensively to perform mathematical operations, manipulate arrays, and access mathematical libraries.

### Input file

The input file is flexible and keyword-based, so you can set as many of the available parameters as you want in any order. The parameters available for customization are as follows:

1. **JX** 
   * X-direction coupling coefficient
   * Units: eV
   * Default value: 0
2. **JY** 
   * Y-direction coupling coefficient
   * Units: eV
   * Default value: 0
3. **JZ** 
   * Z-direction coupling coefficient
   * Units: eV
   * Default value: 0
4. **h\_ext** 
   * External magnetic field coefficient
   * Default value: 0
5. **ext\_dir**
   * Controls direction of external magnetic field (X,Y, or Z)
   * Default value: Z
6. **time\_dep\_flag**
   * Option to have a time-dependent external magnetic field coefficient
     1. Time dependence is implemented via a cosine term
   * Available settings: “y” or “n” without the quotation marks
   * Default value: n
7. **custom\_time\_dep**
   * This provides the option to define a custom time dependence function for the external field term (if time dependence is desired). If this is set to “n”, a standard cosine function will be implemented with the frequency chosen by the “freq” parameter (defined next in this documentation). If this is set to “y”, a custom python function named “external func” must be defined in a python file named “time\_dependence.py” It should only take in “t” as a parameter.
   * Default value: n
   * An example is given below:



1. **freq**
   * If time\_dep parameter is set to “y”, and a custom time dependence function is not specified, this controls the frequency of the cosine term driving the time dependence of the external field
   * Units: 1/fs
   * Default value: 0
2. **delta\_t** 
   * Simulation time step (must be a positive integer)
   * Units: fs
   * Default value: 1
3. **steps**
   * Number of time steps to simulate (must be a positive integer)
   * Default value: 1
4. **shots**
   * Number of shots to execute on the quantum device (must be a positive integer)
   * Default value: 1024
5. **num\_qubits**
   * Number of qubits (must be a positive integer)
   * Default value: 2
6. **initial\_spins**
   * Initial qubit spin vector
   * Qubit state |0> is mapped to spin 1, state |1> is mapped to spin -1
   * Enter vector as list of numbers (1 or -1) separated by commas with no spaces between (e.g. 1,1,1,-1,-1,-1 for a domain wall with 6 qubits)
   * Default value: 1,1
7. **QCQS**
   * Choice between running job on a quantum computer or a quantum simulator
   * Enter “QC” or “QS” without the quotation marks to choose the quantum computer or quantum simulator respectively
   * Default value: QS
8. **device**
   * Which IBM quantum device to run or emulate
     1. See the [IBM Quantum Experience Dashboard](https://quantum-computing.ibm.com/) to see a list of available devices
   * Default value: ibmq\_rome
9. **noise\_choice**
   * This presents the option to include simulated noise if running on a quantum simulator
   * Enter “y” or “n” without quotation marks
   * Default value: n
10. **plot\_flag**
    * This presents the option to output individual plots of the average magnetization of each qubit over time (plots saved as image files)
    * Enter “y” or “n” without quotation marks
    * Default value: y

To set a parameter, simply type \*parameter on one line of the input file, then enter your selection for the parameter on the line below it.

Comments are allowed on any line of the file except the line where the selection for a parameter is entered.

For example: If you’d like to set the number of shots to 2048 and change your IBMQ device to ibmq\_16\_melbourne, you would type:

\*shots #some comment here if desired

2048

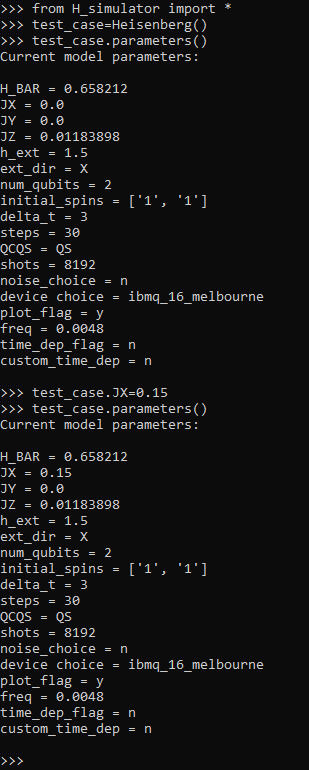
#some arbitrary number of spaces between parameter set statements

\*device

Ibmq\_16\_melbourne

The order in which you set parameters does not matter.

Note: if you would like to change a parameter after creating an instance of the Heisenberg class, you can change it by simply setting “instance\_name.param\_name = new\_value.” An example is shown below:



### The Heisenberg Class and Methods

The Heisenberg class provides a container for everything you’ll need in this software. After creating an instance of the class with a specific input file, or the default one, you can use any of the following methods:

1. **connect\_account**
   1. This connects your IBMQ account to the IBMQ backend, basically starting a session. If you’ve never used the IBMQ system before, you’ll have to call it with an API key found in your profile, but after that no arguments are needed. You only need to call this once within a python session, but there’s also no risks besides longer runtime associated with executing this method before the “run” method every time. See the “run.py” file for an easy example of this latter approach.
2. **parameters**
   1. This prints out the current parameters of the instance
3. **run**
   1. This runs the quantum computer or quantum simulator job described by its associated instance of the Heisenberg object
4. **results**
   1. After executing the “run” method, you can call this method to return the result matrix of qubit average magnetizations over time in numpy array form.

### Connecting to the IBMQ Backend

The directions for configuring an IBMQ account can be found in the directions for the [installation of Qiskit](https://qiskit.org/documentation/install.html), under “Access IBM Quantum Systems.”

When you reach step 4 of the directions above, navigate to line 96 of the Ising input file. Uncomment the line, and paste the API token copied from your IBMQ account into qk.IBMQ.save\_account(‘my\_API\_token’).

After running the software once, you can comment the line back, and uncomment line 98, qk.IBMQ.load\_account(), which will simply load the API token you already have for future experiments.

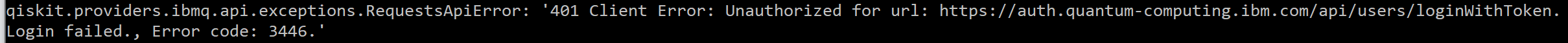
### Running the Software

* + 1. Make sure “H\_simulator.py” and your desired input file(s) are in the same directory
    2. Customize your input file(s) to match your system and outputs of interest
    3. Within a Python script or from the command line, run “from H\_simulator import Heisenberg” or “From H\_simulator import \*”
    4. Initialize an instance of the Heisenberg class with your input file as the sole argument.
    5. Run the “connect\_account” method of that instance
       1. Within any given session, even across multiple instances, you just have to run this method once at the beginning. You can run it every time before running an instance, it will just lengthen the runtime as it overrides the existing session on IBMQ’s backend.
       2. For a simpler but slower workflow, just name your input file “input\_file.txt” and run the “run.py” file from the command line.
    6. Execute the “run” method of the instance
    7. Repeat steps 2, 4, and 6 for as many different systems as desired. Remember, you can also modify existing instances by changing the parameters manually (see the example in Section III)

**Examples are given at the end of this document**

### Troubleshooting

Two Potential Error Messages from IBMQ (and How to Fix)

* + 1. If you get an error that looks something like this:

First, go to the IBM Quantum Experience dashboard and check if you need to accept a new license or terms of agreement. If the problem persists, try modifying line 96 as follows:

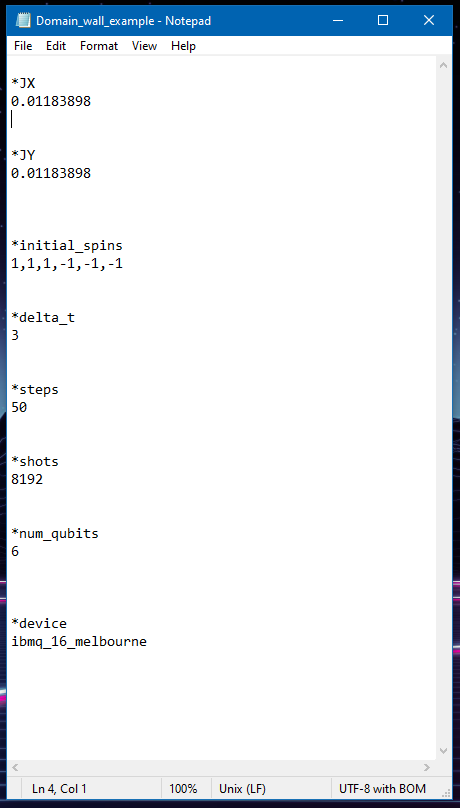
qk.IBMQ.save\_account(‘new\_API\_key’, overwrite=True)

### Examples

#### XX Chain Domain Wall Quench

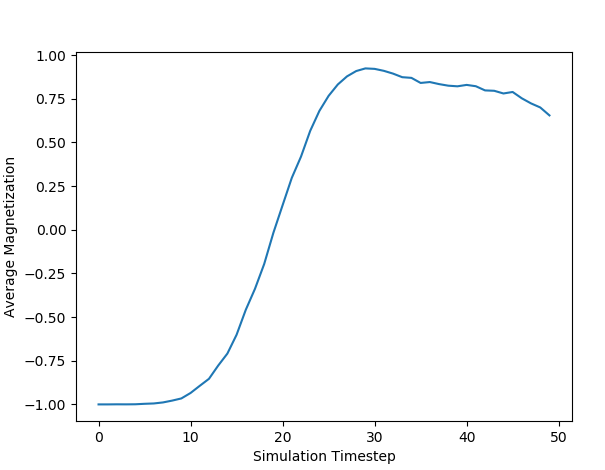
In this example, we will simulate a domain wall quench of a 6 qubit XX spin chain on a noiseless quantum simulator of IBM’s 16-qubit “Melbourne” quantum computer with the goal of plotting the average magnetization of each qubit over time.

The input file would look like this:

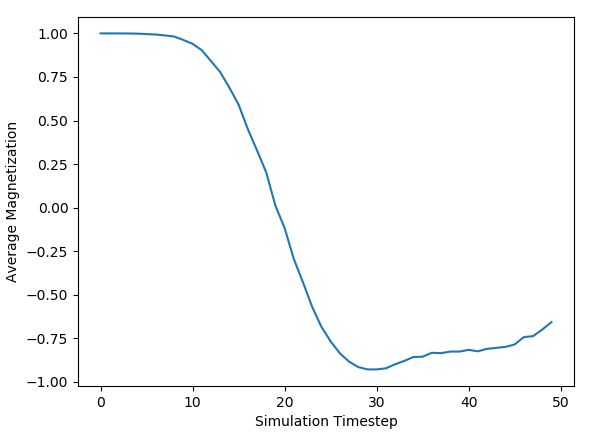


All parameters not listed can be left at their default values for this example. The order in which these parameters are set and the spacing in between the parameters are not important.

This should produce the following evolution for the average magnetization of the first spin in the chain (or the first qubit, in this setup):

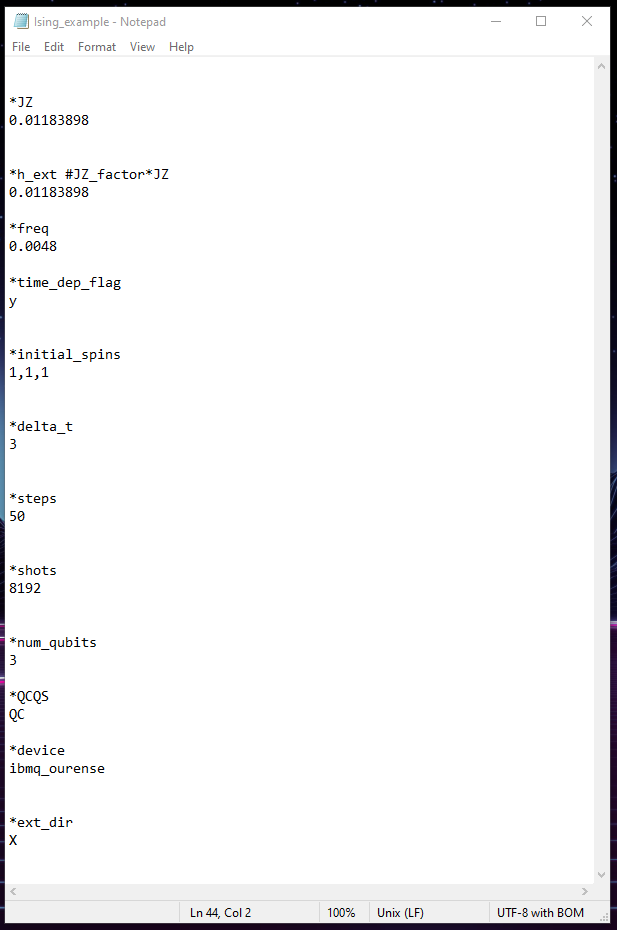


The evolution of the average magnetization of the sixth spin in the chain should look like this:

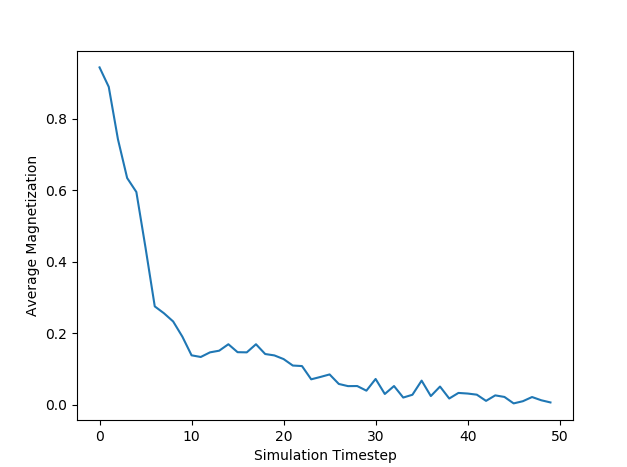


#### Ising Model Example

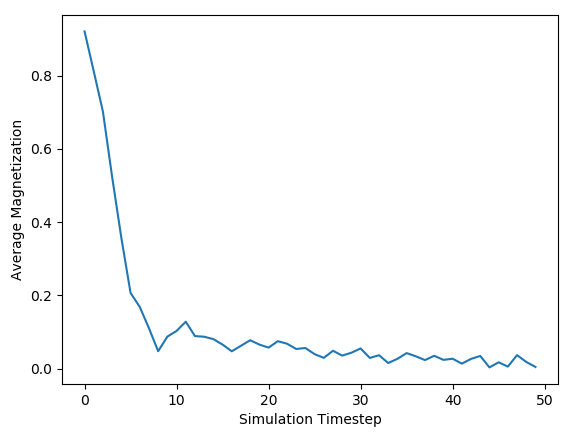
Let’s say we want to simulate an Ising model MoSe2 spin chain of length 3 on a real quantum computer, not a simulator. We will also adopt a standard cosine function for some time dependence in the external magnetic field term. The input file would look like this:



The first spin in the chain should look roughly like the following image, although real quantum computer results will vary from run to run:



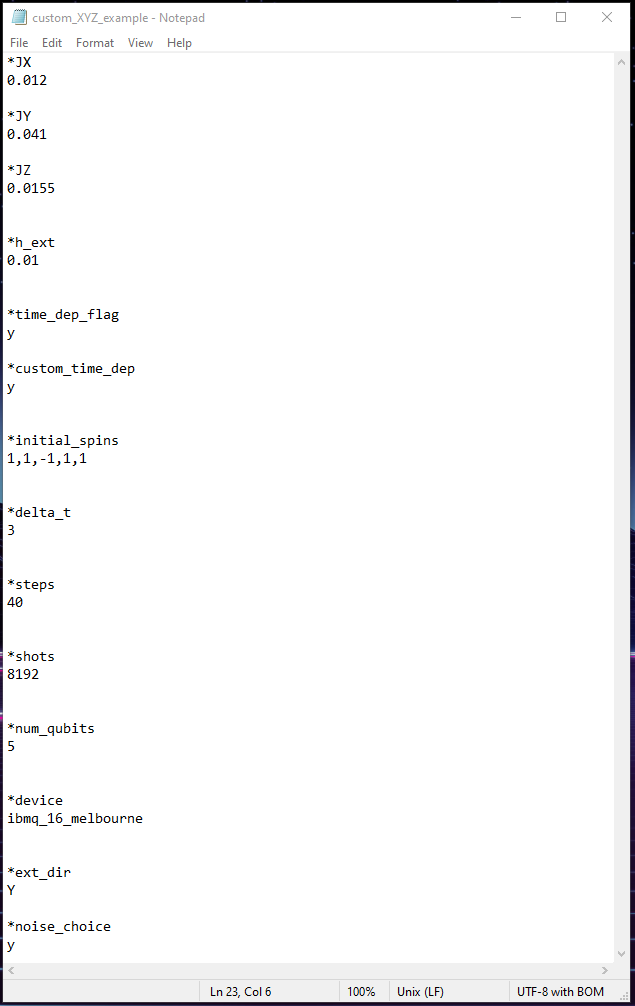
The third spin in the chain should look roughly like this:



#### Custom Time-Dependent XYZ Model Example

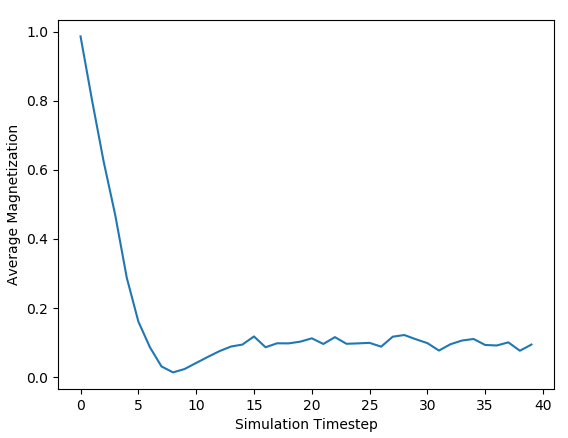
In this example, we will adopt a XYZ subset of the Heisenberg Model for a spin chain of length 5, where JX≠JY≠JZ≠0. We will include an external field term with a custom time dependence function, we will start the third spin in the chain in the “-1” configuration, and we will run this on a noisy quantum simulator of the ibmq\_16\_melbourne quantum computer. This example is primarily to demonstrate the parameter flexibility within the program.

The parameters would be set like this:



Using the default custom time dependence function defined in “time\_dependence.py,” the evolutions of the first and third spins of the chain would resemble the following, respectively:

(again, since this is a noisy simulator, results will vary)



Whereas the third spin in the chain will look something like:

